시스템 프로그래밍

Assignment #1 Finding Sequence

201910162 컴퓨터공학부 문무현

Design and Implementation

실행 시 오류로 처리하고 프로그램을 종료하는 경우는 다음과 같습니다.

1. 인자의 개수(찾을 문자열, 프로세스의 개수)가 정확하지 않은 경우
2. 인자로 들어온 프로세스의 개수가 정수가 아닌 경우
3. 찾을 문자열(argv[1])의 길이가 대상 문자열(input[MAXS])의 길이보다 긴 경우
4. 생성할 자식 프로세스의 개수가 MAXS보다 큰 경우
5. 생성할 자식 프로세스의 개수가 0 이하인 경우
6. 파이프의 생성이 정상적으로 이루어지지 않은 경우

다음의 경우들은 프로그램이 정상적으로 동작하지 않기 때문에 예외로 처리하였습니다. 프로그램이 비효율적으로 동작하더라도 실행에 문제가 없는 경우는 예외로 처리하지 않았습니다. 마지막 문단에 후술하겠습니다.

프로그램 구현의 핵심적인 아이디어는 startingInterval 변수와 dividedLength 변수입니다. startingInterval 변수는 한 프로세스가 이상적으로 나눠 가져야할 문자열의 길이입니다. MAXS / processNum(프로세스의 개수)으로 정의되어 있습니다. 하지만 startingInterval만을 통해 문자열을 분할하는 경우, 찾을 문자열의 사이를 서로 다른 프로세스가 나눠가지는 문제가 발생합니다. 이를 해결하기 위한 방법으로 dividedLength를 따로 정의했습니다. dividedLength 변수는 startingInterval 값에 targetLength(찾을 문자열의 길이)를 더한 값으로 정의됩니다. 자식 프로세스의 분할의 시작점은 startingInteval 변수로 하되, 나눠 가지는 길이는 dividedLength로 했습니다. 따라서 앞뒤의 프로세스들은 찾을 문자열의 길이 만큼씩 중복되는 부분을 가지게 됩니다. startingInteval로 나눴을 때는 찾을 문자열의 사이가 나눠지더라도, dividedLength로 인하여 한 프로세스는 원래 탐색해야 하는 범위에서 targetLength만큼을 추가로 탐색하기 때문에 나눠지는 부분으로 인해 탐색에 실패하는 경우가 발생하지 않습니다. MAXS가 100이고 찾을 문자열의 길이가 5, 생성할 프로세스의 개수가 10개인 경우, startingInterval은 10이고 dividedLength는 15입니다. 따라서 첫번째 자식 프로세스는 대상 문자열의 0~14, 두번째 자식 프로세스는 10~24, 세번째는 20~34 … 로 나누어 가지게 됩니다.

메인 함수에서 자식 프로세스의 생성은 반복문을 통해 인자로 들어온 자식 프로세스의 개수만큼 fork()를 하여 이루어집니다. 이 때 MAXS가 생성할 프로세스의 개수로 나누어 떨어지지 않는 경우를 고려하여 mod 변수를 만들었습니다. mod 변수는 MAXS - processNum(프로세스의 개수) \* startingInterval로 정의됩니다. 즉, MAXS를 processNum으로 나누었을 때의 나머지와 같습니다. 반복문에서 마지막 자식 프로세스를 생성할 때 마지막 자식 프로세스는 다른 프로세스들보다 최대 mod만큼 더 많은 문자열을 가집니다. 대상 문자열의 길이가 100이고 생성할 프로세스의 개수가 8개이며 찾을 문자열의 길이가 2인 경우, startingInteval은 100 // 8인 12이고 dividedLength는 14가 됩니다. 이 때 mod 변수가 없다면, 마지막 자식 프로세스는 84~97까지의 문자열만을 탐색하기 때문에 98, 99번째 문자에 대한 탐색이 이루어지지 않습니다. 이러한 경우를 고려하여 빠지는 부분이 발생하지 않도록 마지막 자식 프로세스는 mod만큼을 추가로 가지도록 했습니다. 마지막 자식 프로세스가 좀 더 많은 부분을 담당하게 되지만, 대상 문자열을 빠짐없이 탐색할 수 있습니다. mod를 더했을 때 마지막 자식 프로세스가 담당하는 부분의 최대 인덱스가 MAXS보다 커질 수도 있지만, 대상 문자열의 마지막에는 항상 널 문자가 존재한다는 조건이 있었으므로 이 부분은 더 이상 고려하지 않았습니다.

대상 문자열을 자식 프로세스들은 strncpy() 문자열 복사 함수를 통해 나눠 가지게 됩니다. 나눠 가질 문자열을 저장할 text[] 변수는 fork() 이전에 선언되었지만, 자식 프로세스들은 서로 다른 메모리 영역을 가지게 되기 때문에 문제가 없습니다. strncpy() 함수가 문자열의 끝에 널 문자를 넣어주지 않기 때문에 text[] 변수는 한 프로세스가 가질 수 있는 최대 길이인 dividedLength + mod에 널 문자를 저장하기 위한 공간 +1 만큼의 크기를 가지도록 선언했습니다. fork()로 생성된 자식 프로세스들은 반복문 안에 존재하며 이전까지의 context 정보를 갖고 있기 때문에, 반복문의 초기식 부분에 선언된 i 변수를 통해 자신이 몇 번째 자식인지 알 수 있습니다. strncpy(text, input + startingInteval \* i, dividedLength)을 통해 대상 문자열에서 각각의 자식 프로세스가 탐색할 부분 문자열들이 정해지게 됩니다.

각각의 자식 프로세스들의 문자열 탐색은 naive 문자열 검색 알고리즘을 통해 이루어집니다. 자신이 나눠 가진 대상 문자열과 찾을 문자열과 일치하는지 단순하게 대상 문자열의 첫번째 인덱스부터 인덱스를 하나씩 증가시키는 방법으로 비교합니다. 탐색에 성공하면 일치하는 부분의 첫번째 인덱스를 파이프에 바로 전송합니다. 이 때 찾은 값은 인덱스에 반복문의 변수 i(즉, 몇 번째 자식 프로세스인지)와 startingInterval을 곱한 값을 더하여 전송합니다. 이 때 더해진 값을 통해서 부모 프로세스는 대상 문자열의 몇 번째 인덱스에서 탐색에 성공했는지 바로 알 수 있습니다. 문자열 탐색이 끝나면 자식 프로세스는 파이프를 닫고 종료합니다.

반복문 안에서 모든 자식 프로세스가 종료되므로 반복문 바깥 부분은 부모 프로세스만 실행하게 됩니다. 여기서 부모 프로세스는 자식 프로세스가 파이프에 전송한 값을 읽습니다. 이 때 부모와 자식 프로세스는 동시에 실행되고 있을 수 있습니다. 따라서 자식 프로세스가 모두 종료되기 전에 부모 프로세스가 모든 fork()를 마치고 파이프를 읽는 영역에 진입하는 경우가 생길 수 있지만, 읽을 데이터가 없는 경우 read()는 block되기 때문에 항상 모든 전송 값을 받을 수 있습니다. 모든 자식 프로세스가 파이프를 닫고 종료한 이후가 되어야 부모 프로세스의 read()가 파이프에 있는 모든 값을 읽고 리턴될 것입니다.

부모 프로세스는 파이프를 통해 자식 프로세스들에게 전송받은 값들은 정렬을 위하여 readArray[] 배열에 저장합니다. 모든 값을 전송받고 파이프를 닫으면 readArray[] 배열에서 선택 정렬을 실행합니다. 오름차순으로 출력할 것이기 때문에 배열에서 가장 작은 값은 선택하여 앞쪽으로 옮기는 과정을 반복하게 됩니다. 출력은 부모 프로세스가 전송받은 값이 존재하는 경우에만 진행됩니다. 정렬된 readArray[] 배열을 반복문을 통해 순서대로 출력합니다. 자식 프로세스들끼리 겹치는 부분이 존재하기 때문에 중복된 값이 전송될 수 있음을 고려하여 출력 시에 배열의 이전 인덱스 값과 비교하여 중복된 값인 경우 출력하지 않도록 했습니다.

해당 방식으로 구현했기 때문에 한 프로세스가 이상적으로 나눠 가져야 할 문자열의 길이보다 탐색 문자열의 길이가 긴 경우, 즉 사용자의 입력에 의해 만들어질 자식 프로세스의 개수가 많아 비효율적인 경우를 예외로 처리할 필요가 없었습니다. MAXS가 100이고 찾을 문자열의 길이가 20, 생성할 프로세스가 10개인 경우가 있을 때, startingInterval은 10, dividedLength는 30이 됩니다. 찾을 문자열과 생성할 프로세스의 개수가 비효율적인 입력이 들어온 상황입니다. 이 경우 첫번째 자식 프로세스는 0~29, 두번째는 10~39에 해당하는 부분을 탐색할 것입니다. 여덟 번째 자식 프로세스는 70~99를 탐색하고, 아홉 번째 자식 프로세스는 80~99를 탐색하며, 열 번째 자식 프로세스는 90~99를 탐색합니다. 이 때 아홉 번째와 열 번째 자식 프로세스는 존재해야 할 이유가 전혀 없으며 오히려 방해가 됩니다. 하지만 사용자가 입력한 문자열과 프로세스 개수의 조합이 비효율적이더라도, 프로그램이 정상적으로 작동할 수 있는 상황이라면 사용자의 요구대로 수행해야 한다고 생각하여 다음과 같이 구현하였습니다.

Function Description

메인 함수에서 모든 기능을 구현했습니다.

|  |  |  |
| --- | --- | --- |
| Function Name | Arguments | Description |
| Main() | int argc | 인자의 개수입니다. |
| char\* argv[] | 인자입니다. 탐색할 문자열과 생성할 프로세스의 개수를 의미합니다. |
| Return Value | 정상 종료 시 0, 비정상 종료 시 -1을 리턴합니다. |